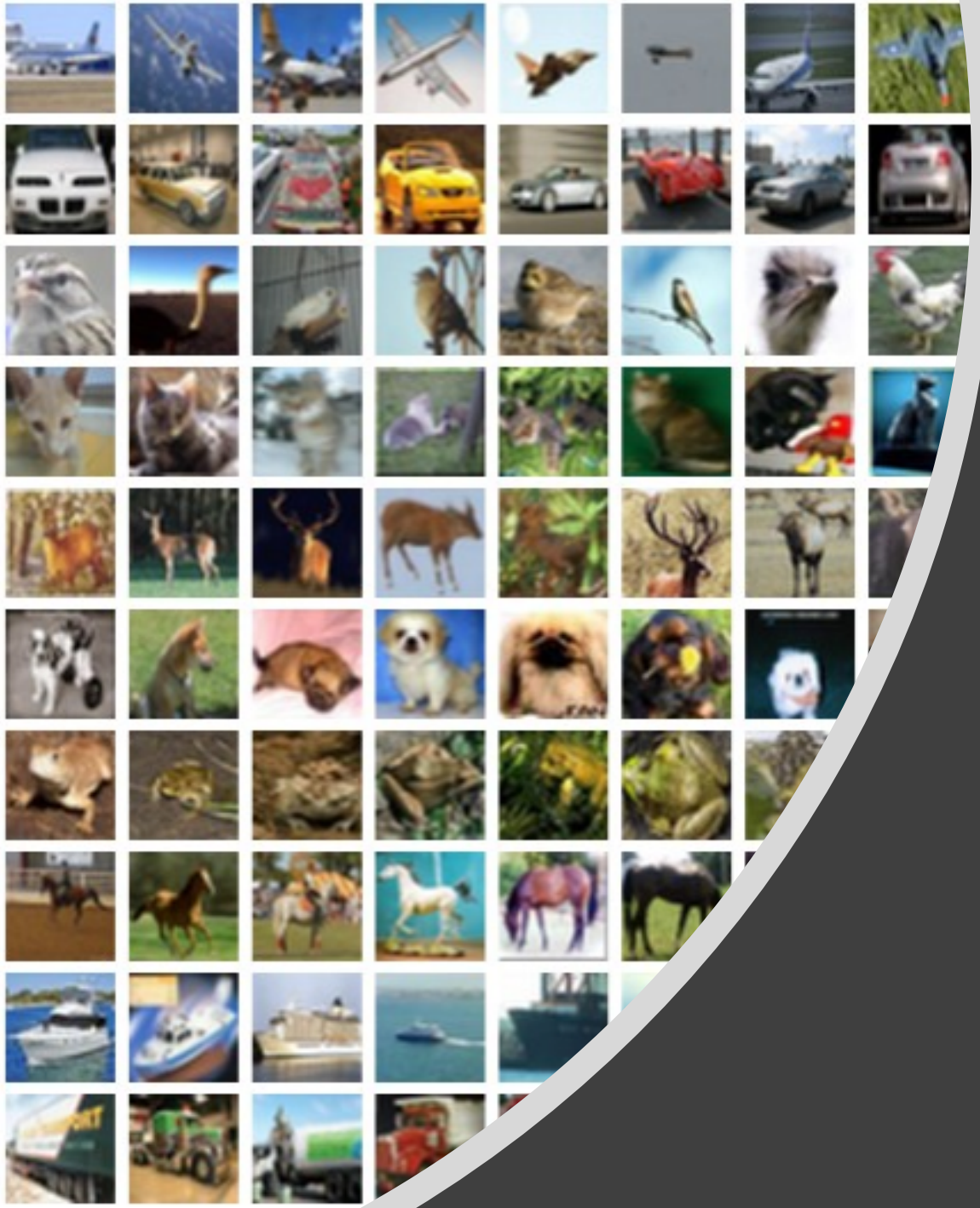
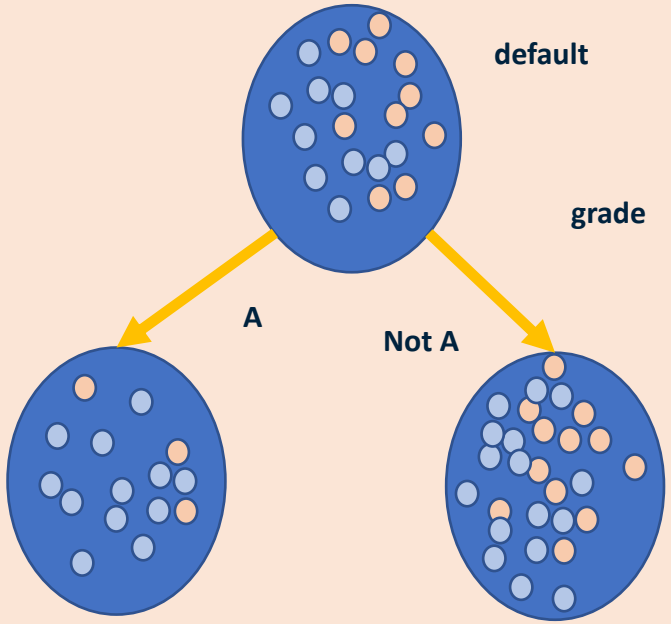


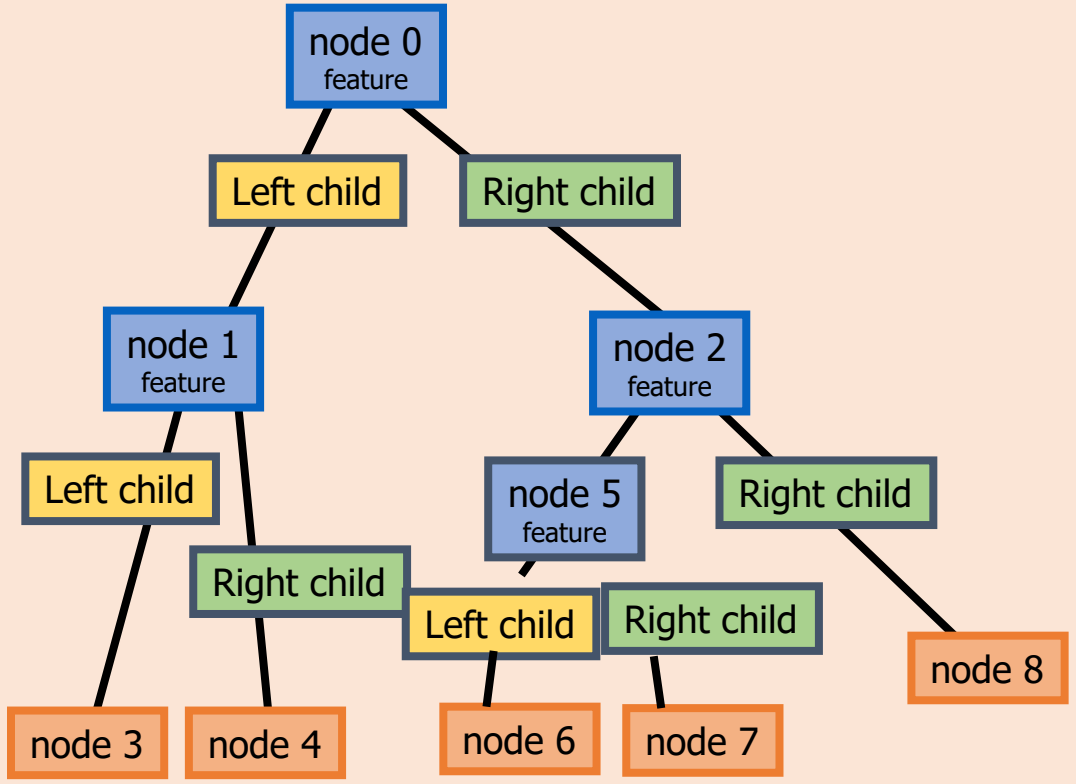
Model Fitting

Classifying Cats

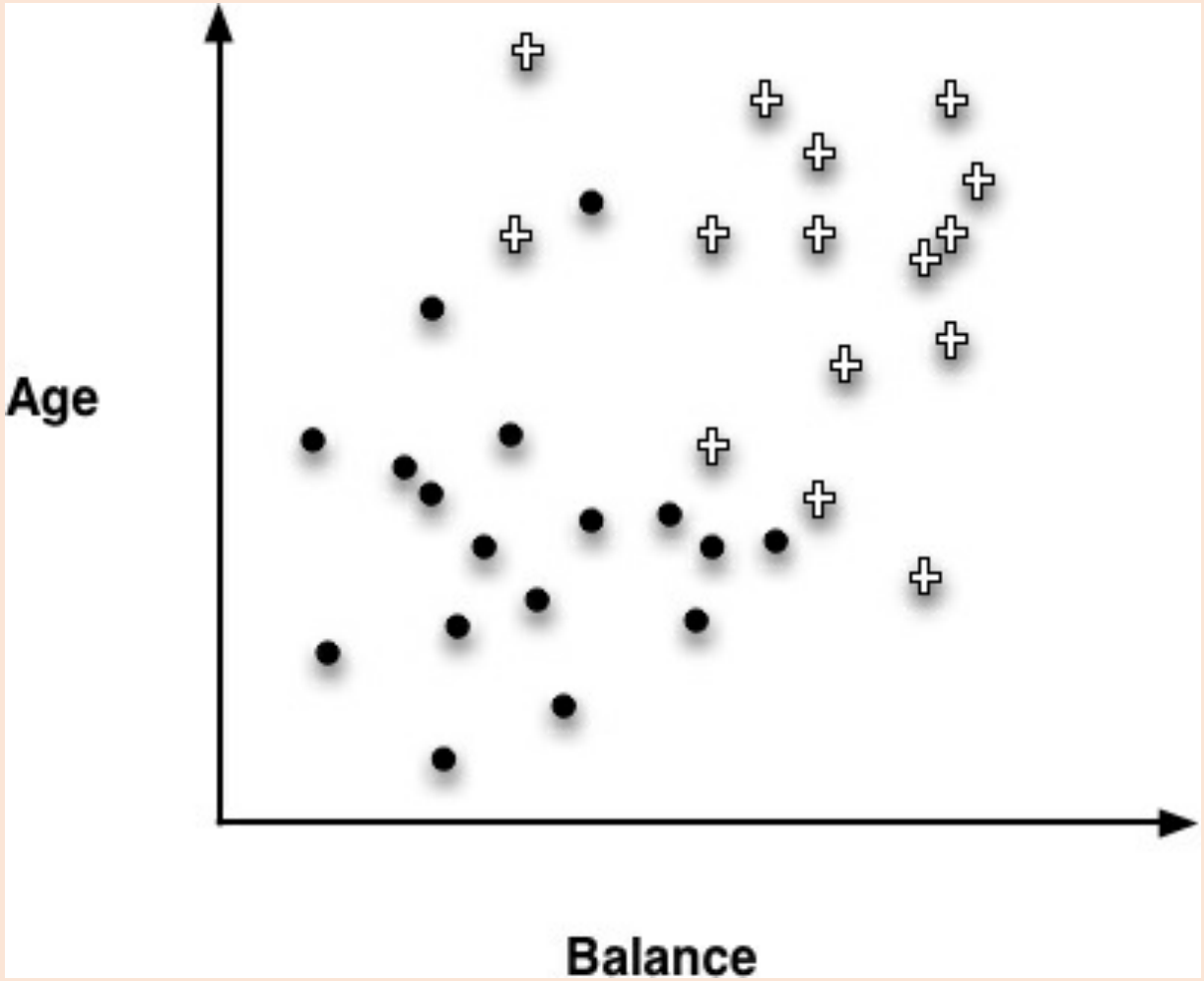
- Decision Tree recap
- Geometry
- Linear models
- Linear Classifiers
- Logistic regression
- Implementation

Decision Tree
Where we left off

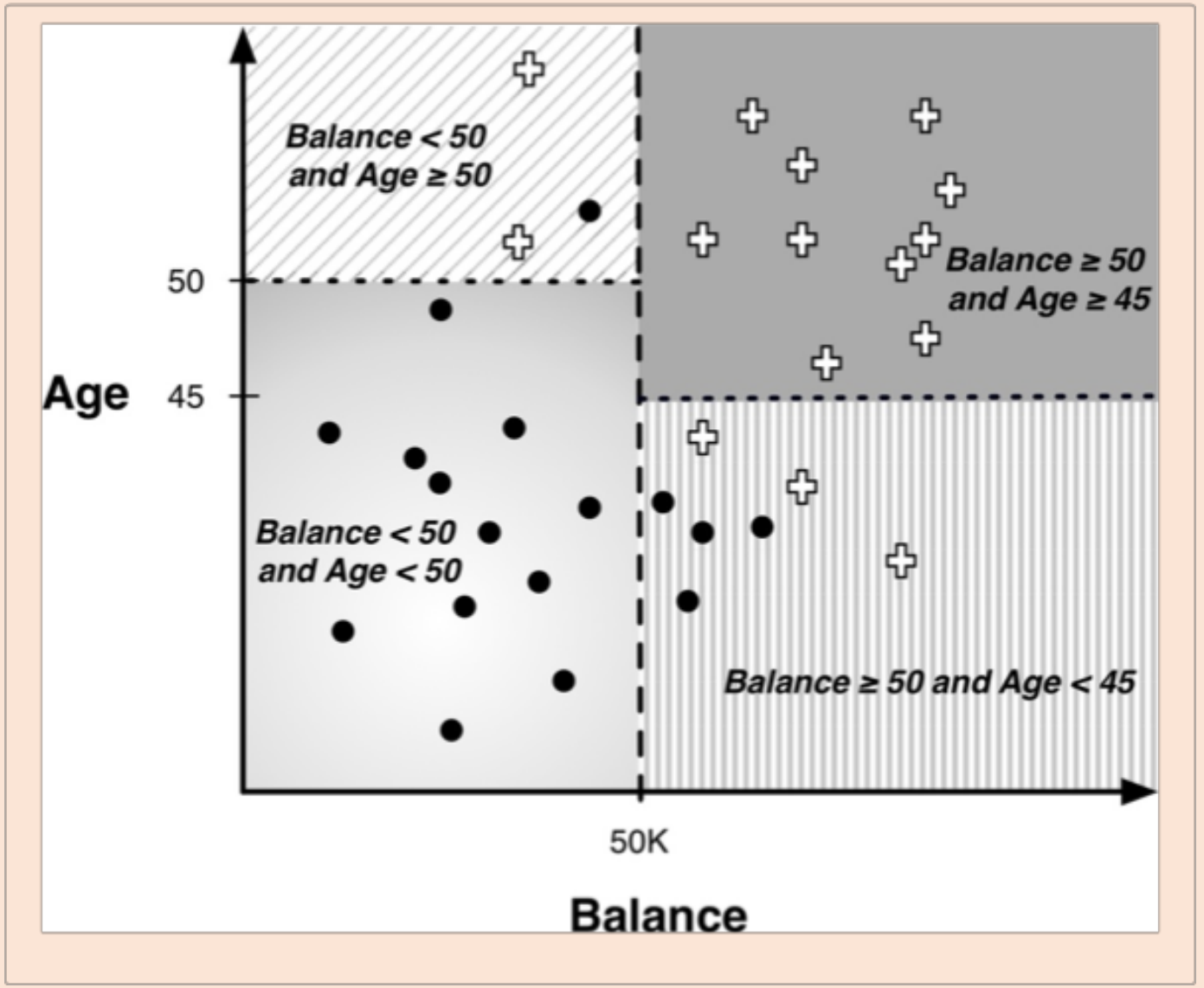




Geometric Interpretation

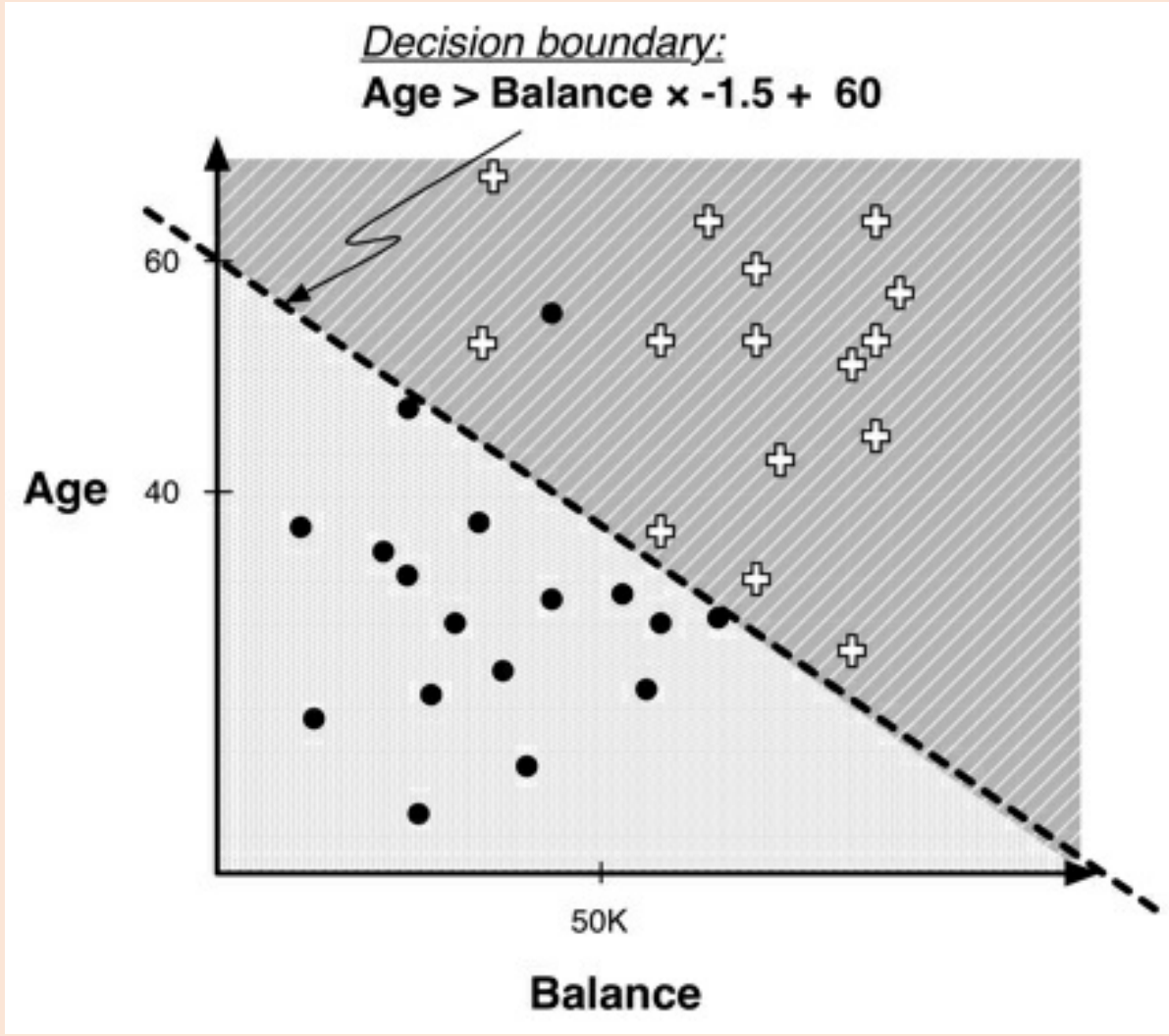


Geometric Interpretation

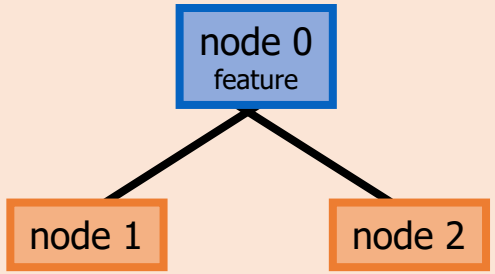


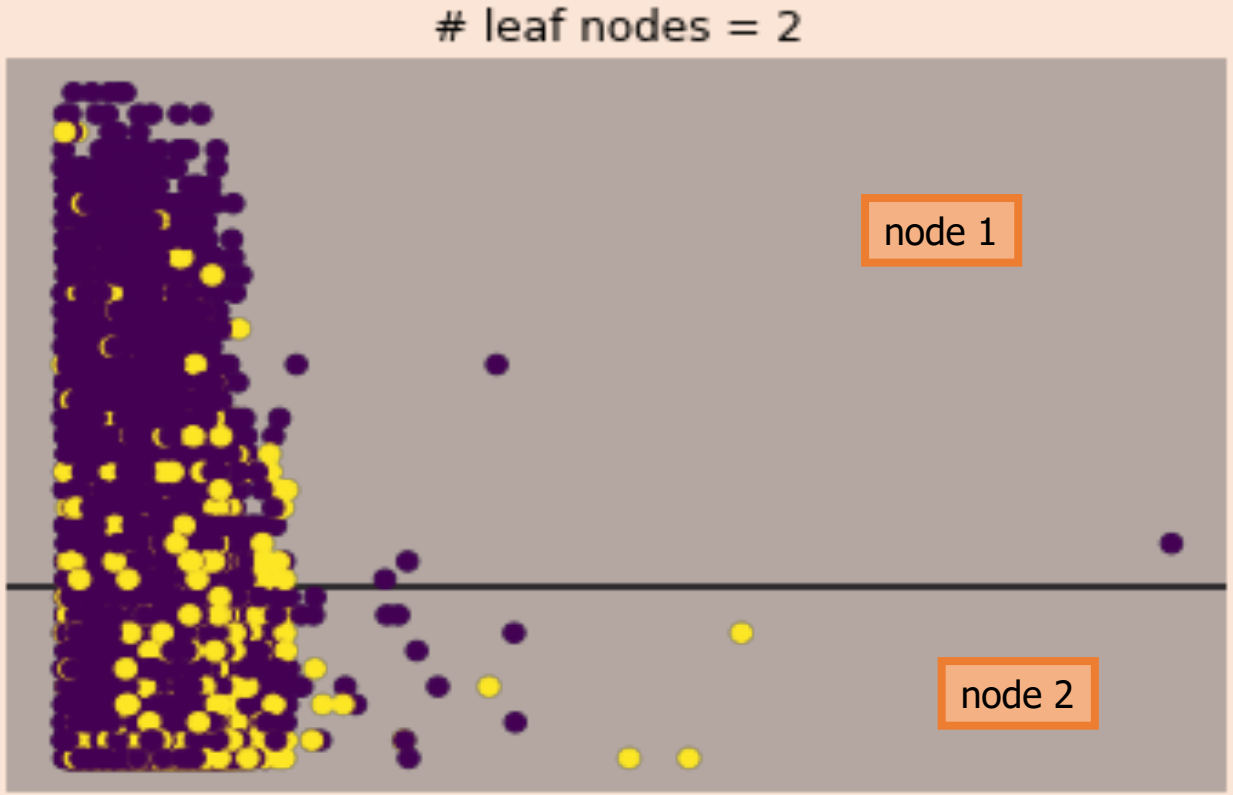
Geometric Interpretation

Linear Regression

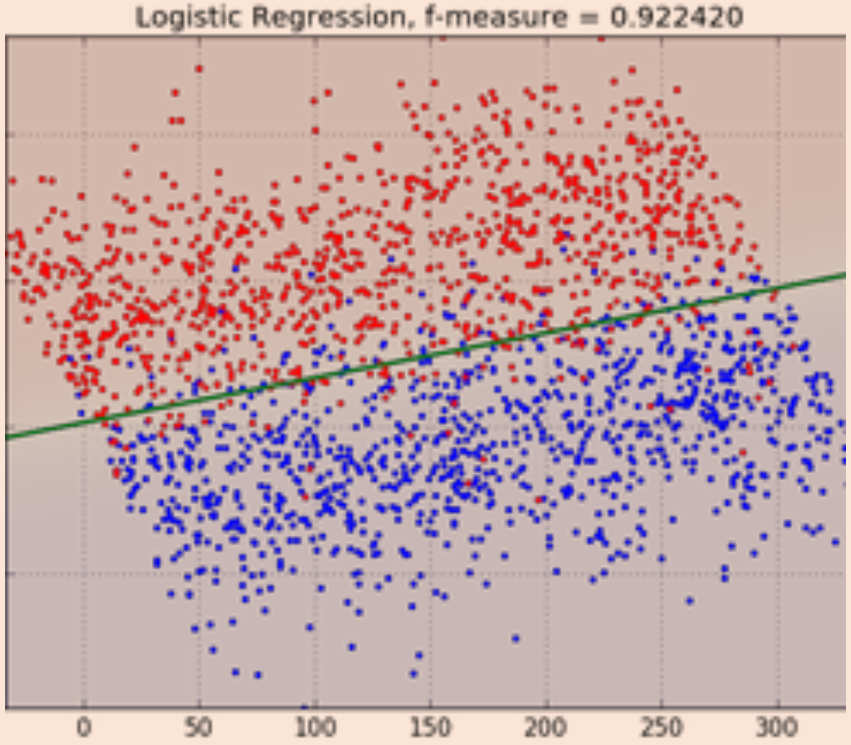


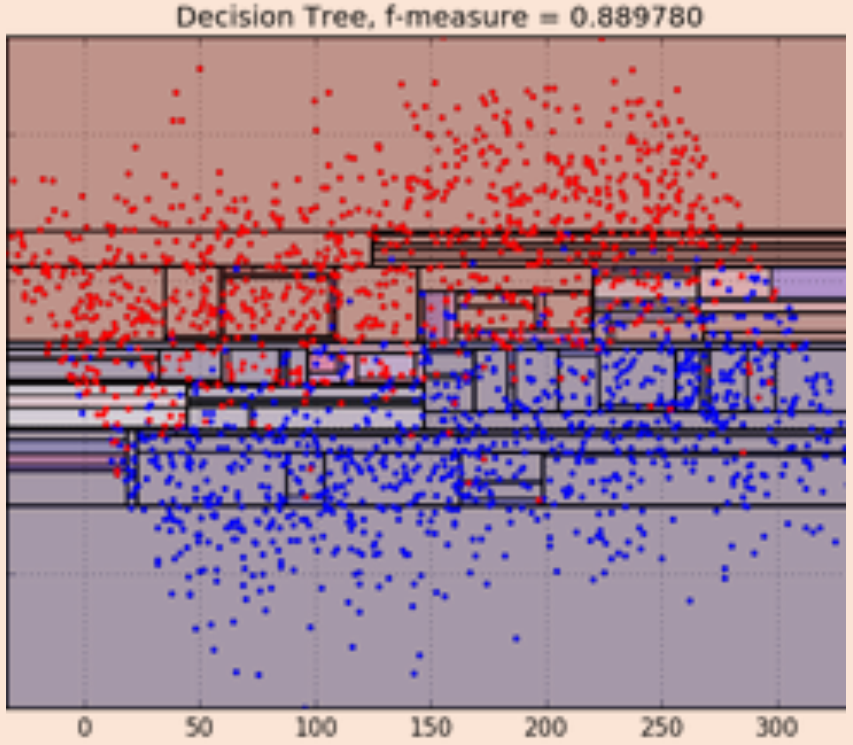
Geometric Interpretation



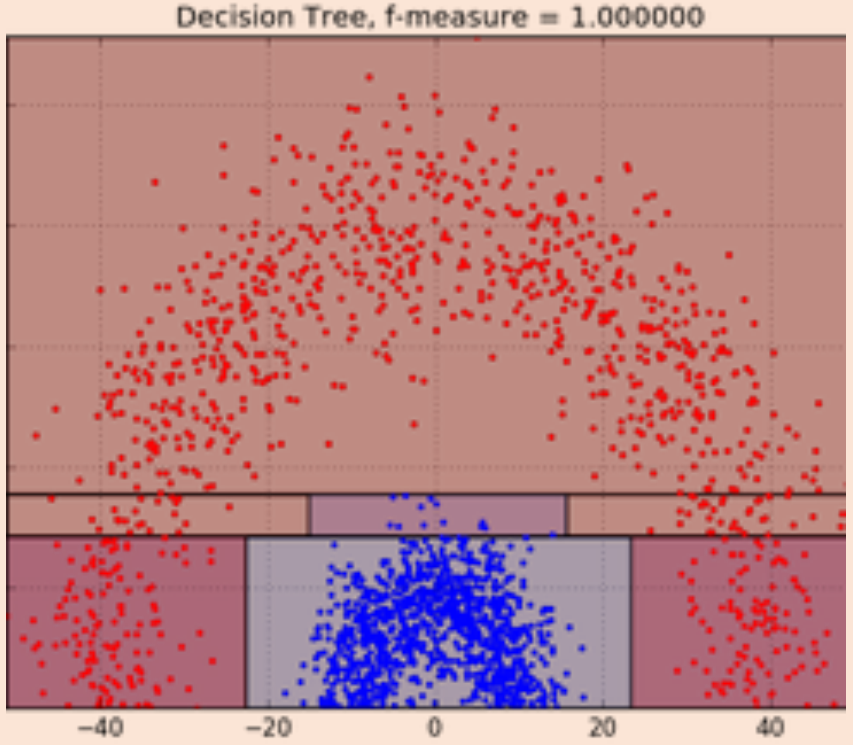


Geometric Interpretation

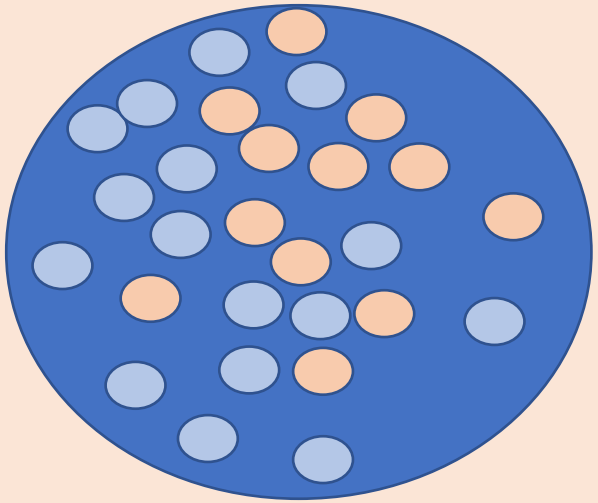




Geometric Interpretation



Parametric and Non parametric



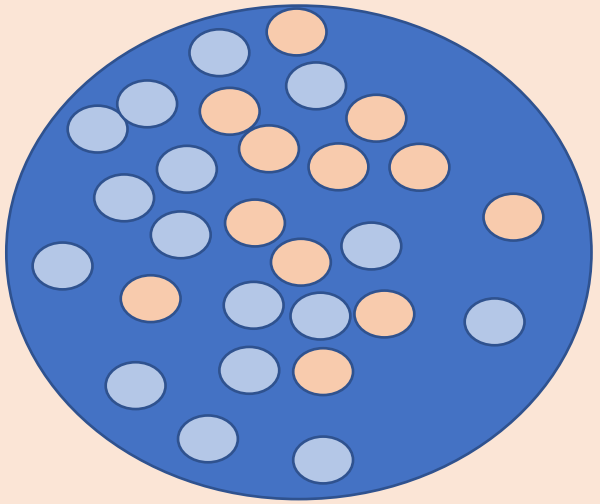
In all cases our objective is to find a model/forecast/function that best fits the data in the sense of minimizing loss

- A **non parametri**c form does not make any assumptions on the functional form
- A **parametric** form assumes a functional form and identifies the parameters that minimize the loss within the category

Loss

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**



Let $\hat{y}_1 \ldots \hat{y}_k$ be as before and p the estimated parameter

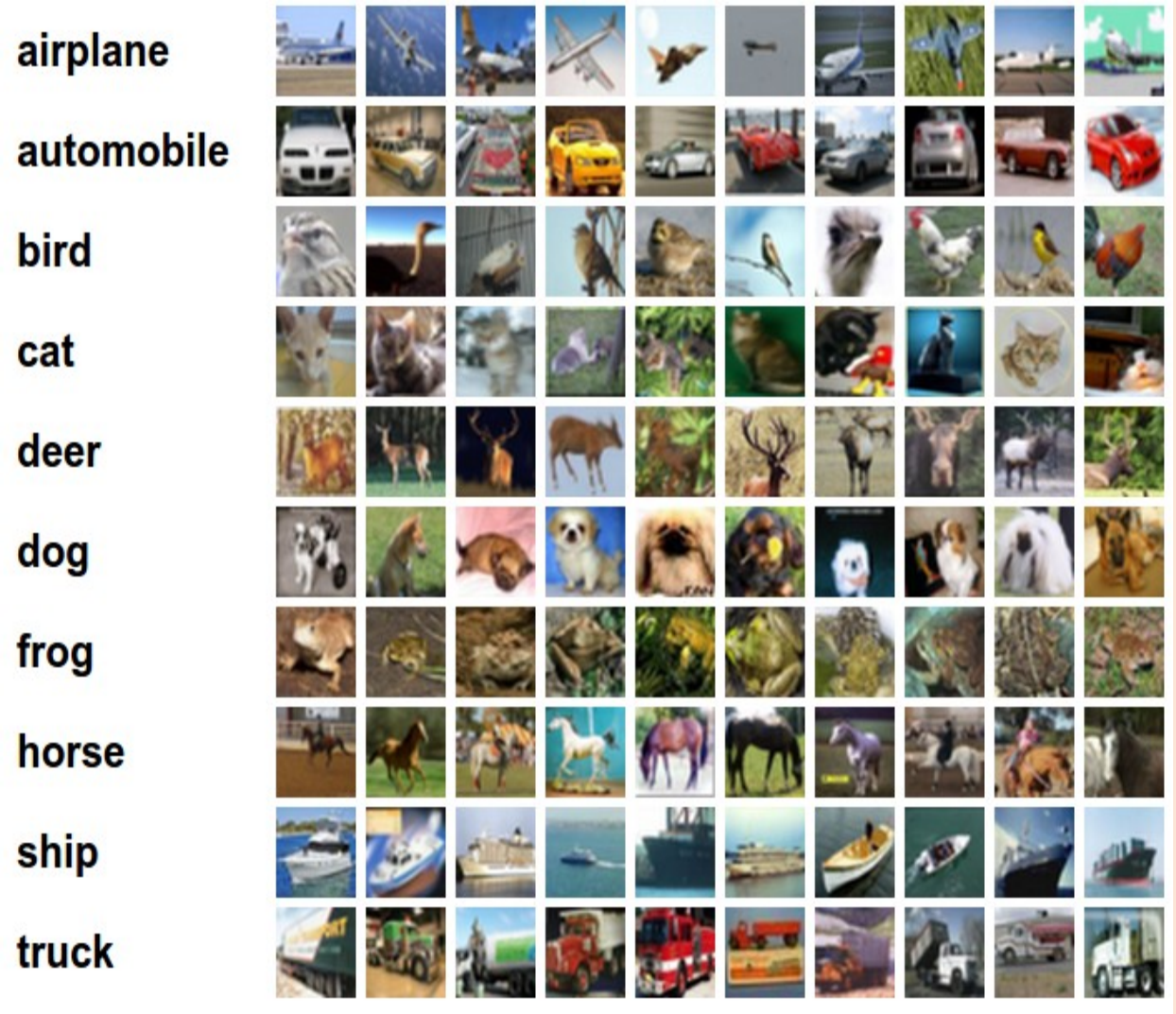
[L_2 - loss]: $L(\hat{y}) = \sum_i (\hat{y}_i - p)^2$

[L_1 - loss]: $L(\hat{y}) = \sum_i |\hat{y}_i - p|$

[0-1 – loss] $L(\hat{y}) = \sum_i \chi_{\hat{y}_i \neq l}$

[likelihood] $L(\hat{y}) = m \cdot log(p) + (k - m) \cdot log(1 - p)$

Image Classification



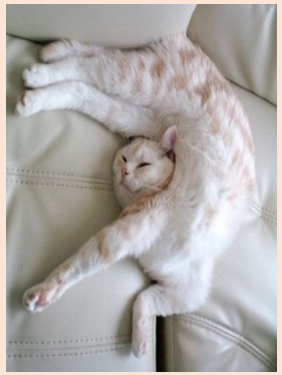
CIFAR-10

- **10** labels
- **50,000** training images
- each image is **32x32x3**
- **10,000** test images.

Image Classification

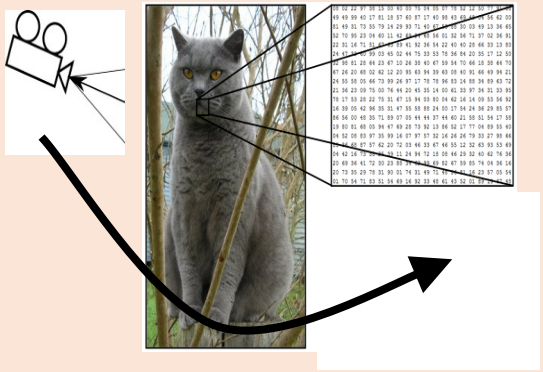
Illumination

Deformation



Occlusion

Camera pose



Background clutter

Intraclass variation

Image Classification

image parameters

$f(\mathbf{x}, \mathbf{W})$

classification

[32x32x3]
array of numbers 0...1

Image Classification

$$f(x, W) = Wx$$

3072x10

10x1

10x3072

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”

Image Classification

$$f(x, W) = Wx$$

10x1

10x 3072

3072x1

(+b) 10x1

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”

Geometric Interpretation

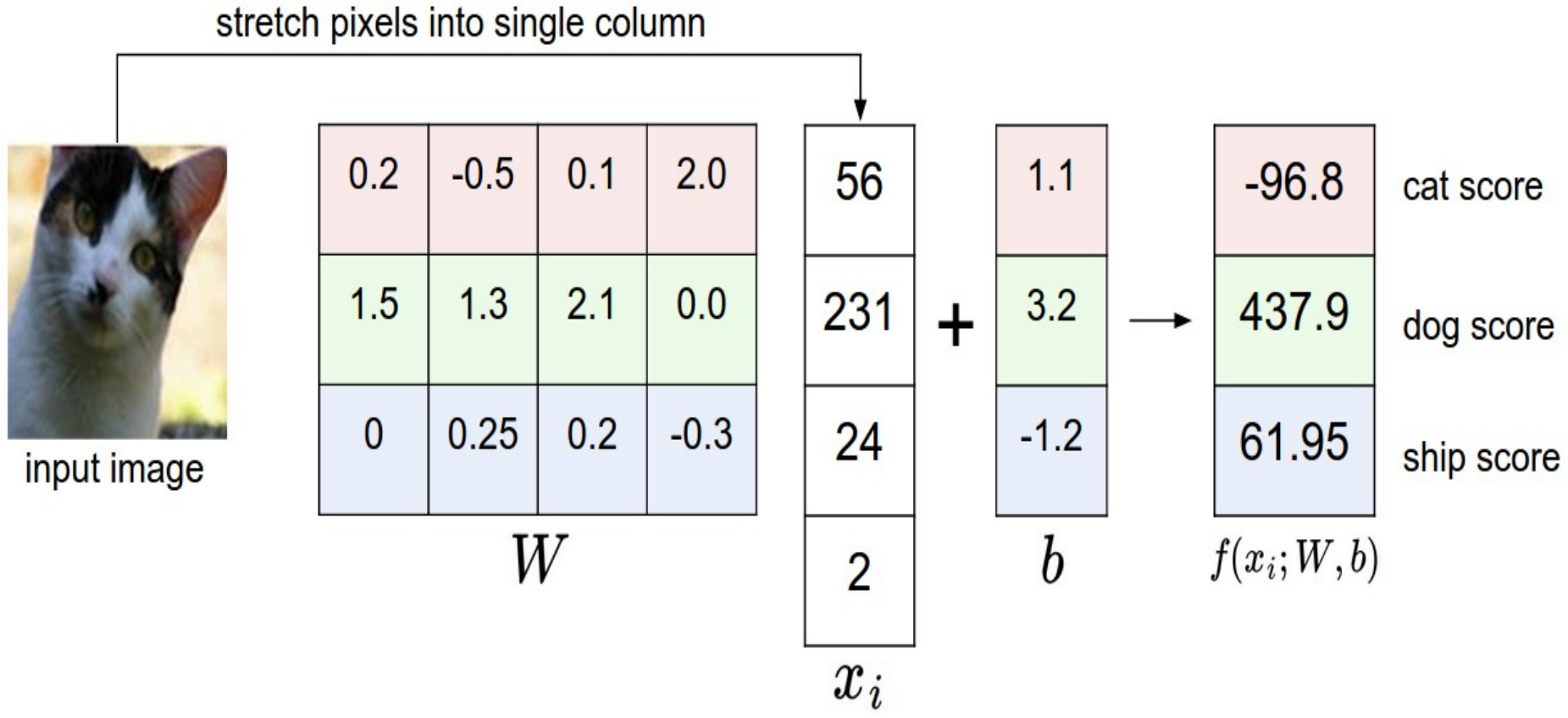


Image Classification

Q: what does the linear classifier do?

$$f(x_i, W, b) = Wx_i + b$$

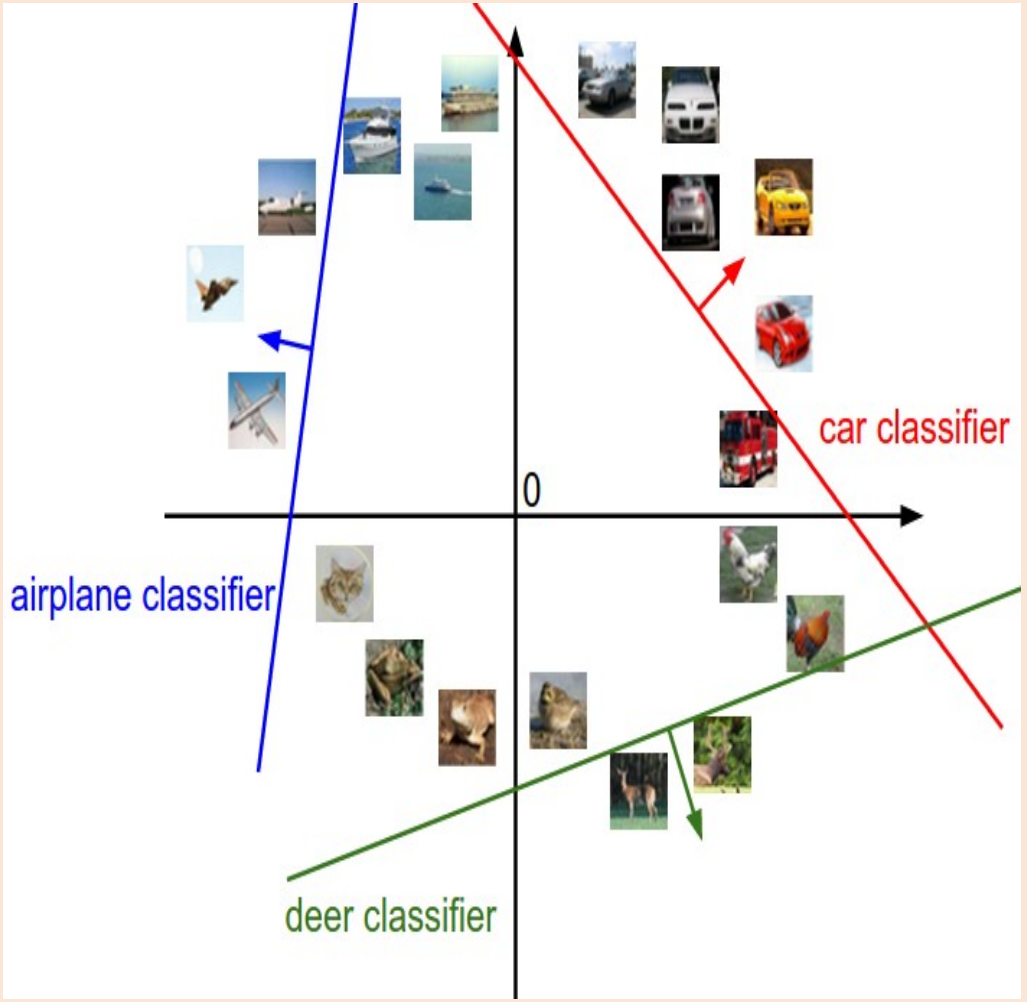
Image Classification

Example trained weights of a linear classifier trained on CIFAR-10:

$$f(x_i, W, b) = Wx_i + b$$

Image Classification

$$f(x_i, W, b) = Wx_i + b$$



[32x32x3]
array of numbers 0...1
(3072 numbers total)

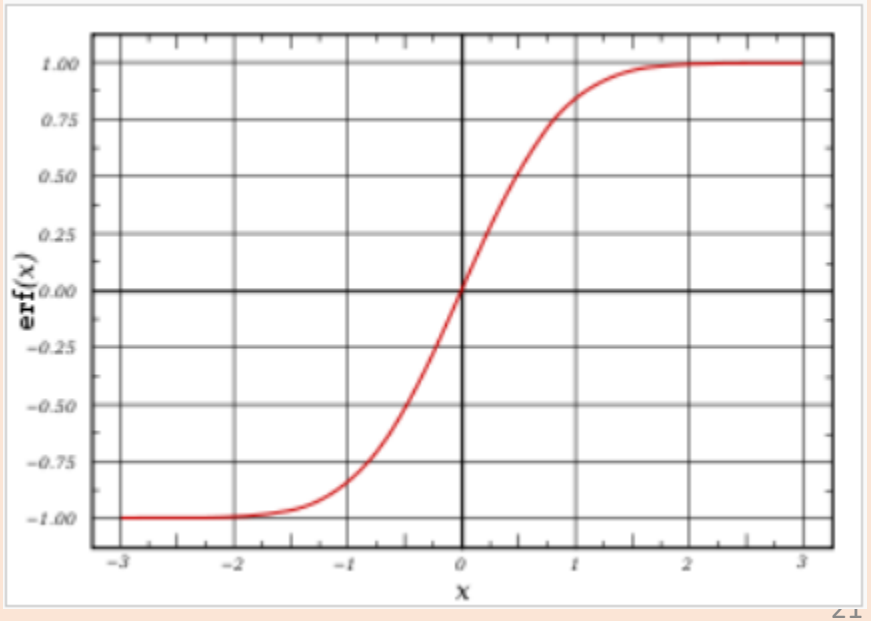
Logistic Regression

Let X be the data instance, and Y be the class label: Learn P(Y|X) directly

Let W = (W1, W2, … Wn), X=(X1, X2, …, Xn), **WX** is the dot product

Sigmoid function:

$$P(Y=1|\mathbf{X}) = \frac{1}{1+e^{-\mathbf{wx}}}$$



Logistic Regression

In logistic regression, we learn the conditional distribution P(y|x)

Let $p_y(x;w)$ be our estimate of P(y|x), where w is a vector of adjustable parameters.

Assume there are two classes, y = 0 and y = 1 and

$$p_0(\mathbf{x};\mathbf{w}) = 1 - \frac{1}{1+e^{-\mathbf{wx}}} \qquad p_1(\mathbf{x};\mathbf{w}) = \frac{1}{1+e^{-\mathbf{wx}}}$$

This is equivalent to $$\log \frac{p_1(\mathbf{x};\mathbf{w})}{p_0(\mathbf{x};\mathbf{w})} = \mathbf{wx}$$

That is, the log odds of class 1 is a linear function of x

22

Learning Algorithm

The conditional data likelihood is the probability of the observed Y values in the training data, conditioned on their corresponding X values. We choose parameters w that satisfy

$$\mathbf{w} = \arg\max_{\mathbf{w}} \prod_{l} P(y^l \mid \mathbf{x}^l, \mathbf{w})$$

where **w** = $<w_0, w_1, \ldots, w_n>$ is the vector of parameters to be estimated, y^l denotes the observed value of Y in the *l* th training example, and $\mathbf{x}^l$ denotes the observed value of **X** in the l th training example

Implementation

In [9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import h5py
import scipy
from scipy import ndimage
from lr_utils import load_dataset
```

Implementation

In [2]:
```
#Load the data (cat/not cat datasets)
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()

#Lets visualize the train set
print(train_set_x_orig)
print(train_set_y)
print(classes)
```

```
[[[[ 17  31  56]
   [ 22  33  59]
   [ 25  35  62]
   ...
   [  1  28  57]
   [  1  26  56]
   [  1  22  51]]

  [[ 25  36  62]
   [ 28  38  64]
   [ 30  40  67]
   ...
   [  1  27  56]
   [  1  25  55]
   [  2  21  51]]

  [[ 32  40  67]
   [ 34  42  69]
   [ 35  42  70]
```

Implementation

In [3]:

```
#Lets get some basic data about our image numpy arrays
m_train = train_set_x_orig.shape[0]
m_test = test_set_x_orig.shape[0]
num_px = train_set_x_orig.shape[1]

print("Number of training examples: m_train = " + str(m_train))
print("Number of test examples: m_test = " + str(m_test))
print("Height/Width of each image: num_px = " + str(num_px))
print("Each image is of size: ("+ str(num_px) + ", " + str(num_px) + ", 3)")
print("train_set_x shape: " + str(train_set_x_orig.shape))
print("train_set_y shape: " + str(train_set_y.shape))
print("test_set_x shape : " + str(test_set_x_orig.shape))
print("test_set_y shape: "+ str(test_set_y.shape))
```

```
Number of training examples: m_train = 209
Number of test examples: m_test = 50
Height/Width of each image: num_px = 64
Each image is of size: (64, 64, 3)
train_set_x shape: (209, 64, 64, 3)
train_set_y shape: (1, 209)
test_set_x shape : (50, 64, 64, 3)
test_set_y shape: (1, 50)
```

Implementation

In [4]:

```
#Will now flatten the numpy array from (num_px, num_px, 3) to (num_px*num_px*3, 1)
#this will make it easier for us so that each image in one numpy array column
train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0], -1).T
test_set_x_flatten = test_set_x_orig.reshape(test_set_x_orig.shape[0], -1).T

print("train_set_x_flatten shape: " + str(train_set_x_flatten.shape))
print("train_set_y shape: " + str(train_set_y.shape))
print("test_set_x_flatten shape: "+ str(test_set_x_flatten.shape))
print("test_set_y shape: "+ str(test_set_y.shape))

#Standardize the dataset for images by dividing each by 255
train_set_x = train_set_x_flatten/255
test_set_x = test_set_x_flatten/255
```

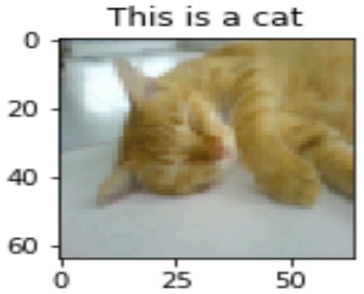
```
train_set_x_flatten shape: (12288, 209)
train_set_y shape: (1, 209)
test_set_x_flatten shape: (12288, 50)
test_set_y shape: (1, 50)
```

Implementation

In [5]:
```
ROWS=64;COLS=64;CHANNELS=3;
classes = {0:'not a cat',
           1:'a cat'}
def show_image(X, y, idx):
    image = X[idx]
    image = image.reshape((ROWS, COLS, CHANNELS))
    plt.figure(figsize=(4,2))
    plt.imshow(image)
    plt.title("This is {}".format(classes[y[0,idx]]))
    plt.show()

def show_image_prediction(X, idx, model):
    image = X[idx].reshape(1, -1)
    image_class = classes[model.predict(image).item()]
    image = image.reshape((ROWS, COLS, CHANNELS))
    plt.figure(figsize=(4,2))
    plt.imshow(image)
    plt.title("Test {}: I think this {}".format(idx, image_class))
    plt.show()
```

In [13]:
```
X=train_set_x_orig
y=train_set_y
show_image(X, y, 121)
print(X[121].shape)
print(X[121])
```




```
(64, 64, 3)
[[[106 102  96]
  [ 90  97 103]
  [104 122 124]
  ...
  [169 148 104]
  [169 147  97]
  [168 145  92]]

 [[117 110  99]
  [108 114 112]
  [119 126 125]
  ...
  [171 149 104]
  [171 149 103]
```

Implementation

In [14]:

```
#We will be using a sigmoid function for our Activation, in Neural Networks most are not ReLU du

def sigmoid(z):
    s = 1/(1+np.exp(-(z)))
    return s

#Create function to set both w and b to 0 to start with
def initialize_with_zeros(dim):
    w = np.zeros((dim,1))
    b = 0
    return w,b
```

Implementation

```
def predict(w, b, X):
    m = X.shape[1]
    Y_prediction = np.zeros((1,m))
    w = w.reshape(X.shape[0],1)
    A = sigmoid(np.dot(w.T,X)+b)
    for i in range(A.shape[1]):
        if A[0,i] <= 0.5:
            Y_prediction[0,i] = 0
        else:
            Y_prediction[0,i] = 1
    return Y_prediction
```